Algorithmen und Datenstrukturen (für ESE) WS 2011 / 2012

Vorlesung 13, Dienstag, 7. Februar 2012
(Editierdistanz, Dynamische Programmierung)

Prof. Dr. Hannah Bast
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

Blick über die Vorlesung heute

- Organisatorisches
  - Ihre Erfahrungen mit dem Ü12 (Dijkstras Algorithmus)
  - Offizielle Evaluation der Vorlesung
- Berechnung der Editierdistanz
  - Definition + Beispiele
  - Lösung mit Rekursion: nicht so gut
  - Lösung mit dynamischem Programmieren: Idee + Algorithmus
  - **Übungsaufgabe (Ü13):** die zweite Lösung implementieren

Ihre Erfahrungen mit dem Ü12 (Dijkstra)

- Zusammenfassung von Ihrem Feedback **Stand 7.2 16:00**
  - Vorlesung und Übungsblatt hat den meisten gut gefallen
  - Für die meisten auch zeitlich sehr gut machbar
  - Dijkstras Algorithmus ist ziemlich interessant
  - Schön, dass man alte Klassen wieder verwenden konnte
  - Vorgaben auf Übungsblatt für Rückgabewerte nicht so gut
  - Kompaktere Beweise wären hilfreicher
  - Prüfungstermin doof, geht auch eine Woche früher?
  - Geben Sie's zu, auch Sie zocken bis spät in die Nacht World of Warcraft und können deshalb noch um halb 2 im Forum antworten! ;-)
  - Herzlichen Glückwunsch zum Google-Preis
  - In Ghana ist es gerade 35°C wärmer
  - C++ Vorlesung nächstes Semester: operator overloading bitte

Ihre Erfahrungen mit dem Ü12 (Dijkstra)

- Fortsetzung ...
  - Einige erst in letzter Minute abgegeben
  - Die hatten dann viele Probleme mit (doofen) Fehlern
  - Warum braucht Dijkstra länger als BFS?
  - Wenn Test abstürzt, keine vernünftige Konsolenausgabe
  - 80 Zeichen pro Zeile ist doof

Offizielle Evaluation der Vorlesung

- Bitte den Bogen **bis Ende der Woche** abgeben
  - Ich würde das Feedback dann nämlich gerne in der letzten Vorlesung zusammenfassen und besprechen
  - Sie bekommen dafür 10 Punkte!
  - Schreiben Sie dazu einfach in Ihre erfahrungen.txt, dass Sie den Evaluationsbogen ausgefüllt haben (wenn es so ist)
  - Nehmen Sie sich bitte genug Zeit für das Ausfüllen
  - Die Freitextkommentare sind für uns am interessantesten
  - Seien Sie bitte ehrlich und möglichst konkret
  - Abgabe elektronisch über das Forum bis spätestens Sonntag Abend, und wenn möglich schon bis Freitag

Editierdistanz — Definition

- Definition Editierdistanz, auch Levenshtein-Distanz
  - Gegeben zwei Zeichenketten (strings) x und y
  - ED(x, y) = Editierdistanz (edit distance) von x und y = die minimale Anzahl Operationen um x in y zu transformieren:
    - Einfügen eines Buchstabens (insert)
    - Ersetzen eines Buchstabens durch einen anderen (replace)
    - Löschen eines Buchstabens (delete)
    - Die Position einer Operation ist ... siehe Beispiel:

D O O F REPLACE(1, B)
B O O F REPLACE(2, L)
B L O F REPLACE(4, E)
B L O E INSERT(5, D)
B L O E D

monoton

B L O E D DELETE(5)
B L O E REPLACE(4, F)
B L O F REPLACE(2, O)
B O O F REPLACE(1, D)
D O O F

nicht monoton, aber es gibt eine monotone ...

Editierdistanz — Eigenschaften

- Etwas Notation
  - Mit ε bezeichnen wir das leere Wort
  - Mit $|x|$ bezeichnen wir die Länge von x (= Anzahl Zeichen)
  - Mit $x[i..j]$ bezeichnen wir die Teilfolge der Zeichen i bis j der Zeichenkette x, wobei $1 \leq i \leq j \leq |x|$
- Ein paar einfache Eigenschaften
  - $ED(x, y) = ED(y, x)$
  - $ED(x, \varepsilon) = |x|$
  - $ED(x, y) \geq abs(|x| - |y|)$ $\quad abs(x) = x > 0 \; ? \; x : -x$
  - $ED(x, y) \leq ED(x[1..n-1], y[1..m-1]) + 1$ $\quad n = |x|, m = |y|$

DOOF BLOED DOO BLOE

Editierdistanz — Lösungsideen

- Wie würden wir als Menschen das Problem lösen
  - für BAUM → MAUER ? 3
  - für MAUER → AMERIKA ? 5
  - für AAEBEAABEAREEAEBA → RBEAAEEBAAAEBBAEAE ?

  → möglichst große gemeinsame Teilstrings zu finden klappt manchmal aber nicht immer
- Rekursiver Ansatz
  - In zwei Teile / Hälften teilen? Keine gute Idee, z.B.
    - ED(GR|AU, RA|UM) = 2 aber
    - ED(GR, $\underbrace{\text{RA)}}_{=2}$ + ED(AU, $\underbrace{\text{UM)}}_{=2}$ = 4
  - Auf ein "kleineres" Problem zurückführen?
    - das probieren wir jetzt

Editierdistanz — Rekursiver Ansatz 1/3

SEHRDOOF
SEHROOF
DELETE (5)
→ SEHR

- Erst noch etwas Terminologie
  - Seien x und y unsere beiden Zeichenketten
  - Seien $\sigma_1, \ldots, \sigma_k$ eine Folge von $k = ED(x, y)$ Operationen für $x \rightarrow y$, das heißt um x in y zu überführen

    (Wir nehmen im Folgenden nicht an, dass wir die Folge schon kennen, sondern nur, dass es so eine gibt)
  - Wir betrachten im Folgenden nur **monotone** Op.-Folgen, d.h. die Position von σ_{i+1} ist $\geq$ die Position von σ_i, wobei $=$ nur dann erlaubt ist, wenn beides delete Operationen sind
  - **Lemma:** Für beliebige x und y mit $k = ED(x, y)$ gibt es eine **monotone** Folge von k Operationen für $x \rightarrow y$
  - Beweisintuition: die Reihenfolge der Operationen ist im Grunde egal, also kann man sie auch monoton anordnen

Editierdistanz — Rekursiver Ansatz 2/3

BLOED DOOFD DOOFD DOOF
1b: $x \xrightarrow[3]{} z$, $z \xrightarrow[1]{} y$

BLÖD DOOD DOOD DOOF
1c: $x \xrightarrow[3]{} z$, $z \xrightarrow[1]{} y$

- Fallunterscheidung
  - Wir betrachten die letzte Operation σ_k
    - DOOF BLOE BLOE BLOED
    - $\sigma_1, \ldots, \sigma_{k-1} : x \rightarrow z$ und $\sigma_k : z \rightarrow y$
    - Seien $\underset{4}{n} = |x|$ und $\underset{5}{m} = |y|$ und $\underset{4}{m'} = |z|$
    - Man beachte, dass $m' \in \{m - 1, m, m + 1\}$ wieso?
  - Fall 1: σ_k macht etwas "ganz am Ende" von z, d.h. eins von:
    - Fall 1a: σ_k = insert(m' + 1, y[m]) [dann ist $m' = m - 1$]
    - Fall 1b: σ_k = delete(m') [dann ist $m' = m + 1$]
    - Fall 1c: σ_k = replace(m', y[m]) [dann ist $m' = m$]
  - Fall 2: σ_k macht nichts "ganz am Ende" von z
    - dann $z[m'] = y[m]$ und $x[n] = z[m']$ und damit

      $\sigma_1, \ldots, \sigma_k : x[1..n-1] \rightarrow y[1..m-1]$ und $x[n] = y[m]$

BLÖDI DOODI DOODI DOOFI
2: $x \xrightarrow[3]{} z$, $z \xrightarrow[1]{} y$

Editierdistanz — Rekursiver Ansatz 3/3

- Wir haben also einen dieser vier Fälle
  - Fall 1a (insert am Ende): $\sigma_1, \ldots, \sigma_{k-1} : \overset{\text{DOOF}}{x[1..n]} \rightarrow \overset{\text{BLOE}}{y[1..m-1]}$
  - Fall 1b (delete am Ende): $\sigma_1, \ldots, \sigma_{k-1} : \overset{\text{BLOE}}{x[1..n-1]} \rightarrow \overset{\text{DOOF}}{y[1..m]}$
  - Fall 1c (replace am Ende): $\sigma_1, \ldots, \sigma_{k-1} : \overset{\text{BLÖ}}{x[1..n-1]} \rightarrow \overset{\text{DOO}}{y[1..m-1]}$
  - Fall 2 (nichts am Ende): $\sigma_1, \ldots, \sigma_k : \overset{\text{BLÖD}}{x[1..n-1]} \rightarrow \overset{\text{DOOF}}{y[1..m-1]}$
- Daraus folgt die rekursive Formel
  - Für $|x| > 0$ und $|y| > 0$ ist $ED(x, y)$ = das Minimum von
    - $ED(x[1..n], y[1..m-1]) +1$, und
    - $ED(x[1..n-1], y[1..m]) + 1$, und
    - $ED(x[1..n-1], y[1..m-1] + 1$ falls $x[n] \neq y[m]$
    - $ED(x[1..n-1], y[1..m-1]$ falls $x[n] = y[m]$
  - Für $|x| = 0$ ist $ED(x, y) = |y|$, für $|y| = 0$ ist $ED(x, y) = |x|$

Editierdistanz — Beispielberechnung

	ε	B	L	O	E	D
ε	0	1	2	3	4	5
D	1	1	2	3	4	4
O	2	2	2	2	3	4
O	3	3	3	2	3	4
F	4	4	4	3	3	4

der Eintrag enthält ED(DO, BLO)

das ist ED(DOOF, BLOED)

keine Op. nötig

DOOF	INSERT(1, B)	1
BDOOF	REPLACE(2, L)	2
BLOOF	REPLACE(4, E)	3
BLOEF	REPLACE(5, D)	4
BLOED		

es gibt mehrere optimale Folgen

$\Rightarrow$ minimale?

Editierdistanz — Folge von Operationen

- Die Tabelle gibt uns auch eine optimale Folge
  - Wir merken uns einfach bei jeder Anwendung der Rekursionsformel, welcher der vorherigen Einträge den kleinsten Wert ergeben hat (die Pfeile in unserem Bild)
    - Es kann von einem Eintrag mehrere Pfeile zu den drei Einträgen davor geben
  - Wenn wir den Pfeilen von dem Eintrag bei (n, m) bis zum Eintrag für (0, 0) folgen, bekommen wir eine optimale Folge von Operationen
    - Können wir unterwegs mehreren Pfeilen folgen, gibt es entsprechend mehrere optimale Folgen
    - Diese Folgen sind nach Konstruktion alle monoton

Editierdistanz — Programm

$\Rightarrow T(n,n) = T(n-1,n) + T(n,n-1)$
$+ T(n-1,n-1) + 1$
$\geq 3 \cdot T(n-1,n-1)$

- Rekursives Programm
  - Es liegt nahe, das rekursiv zu programmieren
  - Für die Laufzeit würde folgende rekursive Formel gelten

    $T(n, m) = T(n-1, m) + T(n, m-1) + T(n-1, m-1) + 1$
  - Man kann leicht ausrechnen, dass dann $T(n, n) \geq 3^n$
  - Das heißt die Laufzeit wäre (mindestens) exponentiell
- Dynamische Programmierung
  - Wir berechnen die Tabelle einfach Eintrag für Eintrag, so wie wir es in dem Beispiel eh gemacht haben und merken uns alle Einträge, die wir schon berechnet haben
  - Das braucht dann Laufzeit und Speicherplatz $O(n \cdot m)$

Dynamisches Programmieren

- Allgemeines Prinzip
  - Reduziere das Problem auf Unterprobleme kleinerer Größe (bei der Editierdistanz: kleineres |x| + |y|)
  - Für eine gegebene Eingabe berechne alle Unterprobleme in der Reihenfolge aufsteigender Größe
  - Die Laufzeit und der Platzverbrauch hängen dann von der Anzahl dieser Unterprobleme ab (bei der ED: |x| · |y|)
  - Zusammen mit dem "Wert" der optimalen Lösung erhält man auch leicht einen "Weg" dorthin (wie bei Dijkstras Alg. auch)
- Warum haben wir z.B. QuickSort nicht so realisiert?
  - Da brauchten wir nicht die Lösung aller möglichen Unterprobleme, sondern eine beliebige Aufteilung in zwei Teile war gut genug; bei der ED müssen wir quasi "alles ausprobieren"

Literatur / Links

- Dynamische Programmierung
  - In Mehlhorn/Sanders:
    12.3 Dynamic Programming
  - In Wikipedia
    http://en.wikipedia.org/wiki/Dynamic_programming
    http://de.wikipedia.org/wiki/Dynamische_Programmierung
- Editierdistanz
  - In Wikipedia
    http://en.wikipedia.org/wiki/Levenshtein_distance
    http://de.wikipedia.org/wiki/Levenshtein-Distanz